INT. J. ELECTRONICS, 1990, VOL. 69, NO. 4, 509–518

Efficient algorithm for Weinberger array folding

SADIQ M. SAIT† and
MUHAMMAD ABDUL-AZIZ AL-RASHED†

Weinberger arrays (WAs) are an alternative to programmable logic arrays (PLAs) as a method of implementing combinational logic circuits. Column folding is a technique generally employed in PLAs to save the silicon area by implementing the circuit of two columns in the area of one. This paper describes an algorithm for folding Weinberger arrays. The algorithm performs column and row ordering and then column folding. The complexity of the algorithm is $O(n^2)$. The algorithm has been tested extensively on WAs that were generated both for practical circuits and for numerous randomly generated ciruits. Experimental results are discussed.

1. Introduction

A Weinberger array (WA) (Weinberger 1967) is a logic array that is similar to a programmable logic array (PLA) (Ayres 1983). It differs from PLA in that it only consists of one plane. Weinberger arrays can take several forms, one of which is the NOR structure, where there is a single rectangular plane consisting of only NOR gates (and Inverters).

WAs are an alternative to PLAs as a method of implementing combinational logic circuits (Weinberger 1967). They have certain advantages over PLAs, in that their area does not generally grow as fast as that required for PLAs as the size of the problem grows. If the final implementation is in n-MOS technology, then large PLAs require large widths for power and ground lines (in metal) to avoid the problem due to metal migration (Mead and Conway 1980, Ayres 1983). The other advantage of the WAs is that, unlike PLAs, they accommodate forms of logic other than the standard two-level sum of products (SOP). Functions expressed as an arbitrarily deep SOP, that is, multilevel functions, can be easily realized using WAs.

WAs have certain disadvantages. One is that the optimal design problem is considerably more complex than with PLAs. The other disadvantage is that WAs tend to take an awkward shape, that is, being very much wider than high. However, by folding, the width of the WA is reduced considerably.

As an example consider the two functions below:

$$s = c(x \cdot y + \bar{x} \cdot \bar{y}) + \bar{c}(\bar{x} \cdot y + x \cdot \bar{y})$$

$$d = (\bar{x} + \bar{y})' + (\bar{c} + (x + y)')'$$

To implement the above expressions using a NOR Weinberger array the function is re-expressed in standard NOR form as shown below.

$$s = (\bar{c} + ((\bar{x} + \bar{y})' + (x + y)')')' + (c + ((x + \bar{y})' + (\bar{x} + y)')')'$$

$$s = (\bar{c} + (I + J)')' + (c + (L + M)')'$$

$$s = (\bar{c} + N)' + (c + O)'$$

$$s = (P + Q)$$

$$s' = (P + Q)'$$

Received 27 July 1989; accepted 10 October 1989.

† Department of Computer Engineering, KFUPM 673, Dhahran-31261, Saudi Arabia.

$$d = (\bar{x} + \bar{y})' + (\bar{c} + (x + y)')'$$

$$d = I + (\bar{c} + J)'$$

$$d = (I + K)$$

$$d' = (I + K)'$$

In the above example, the external inputs are x, $\bar{x}$, y, $\bar{y}$, c and $\bar{c}$, and the internal inputs are $I = \text{NOR}(\bar{x}, \bar{y})$, $J = \text{NOR}(x, y)$, $K = \text{NOR}(\bar{c}, J)$, $d' = \text{NOR}(I, K)$, $L = \text{NOR}(x, \bar{y})$, $M = \text{NOR}(\bar{x}, y)$, $N = \text{NOR}(I, J)$, $O = \text{NOR}(L, M)$, $P = \text{NOR}(\bar{c}, N)$, $Q = \text{NOR}(c, O)$, and $s' = \text{NOR}(P, Q)$.

The n-MOS stick diagram (Mead and Conway 1980) of the Weinberger array in mixed notation is given in Fig. 1. Column folding is a technique generally employed in PLAs (Brayton *et al.* 1982, Hemachandra 1982) to save the silicon area by implementing the circuit of two columns in the area of one. This paper describes an efficient algorithm for column folding of Weinberger arrays. The algorithm performs column and row ordering and then column folding.

In § 2 we define a Weinberger personality. The column-folding algorithm is presented in § 3. Experimental results and concluding remarks are given in § 4. Pseudo-code and actual detailed algorithms for ordering (ORDER) and folding (FOLD) are given.

2. Weinberger personality

The columns of NOR WA in n-MOS run in metal. A contact cut connects the columns to the row. The row runs horizontally in polysilicon and carries the output of the column to the inputs of other gates (columns).

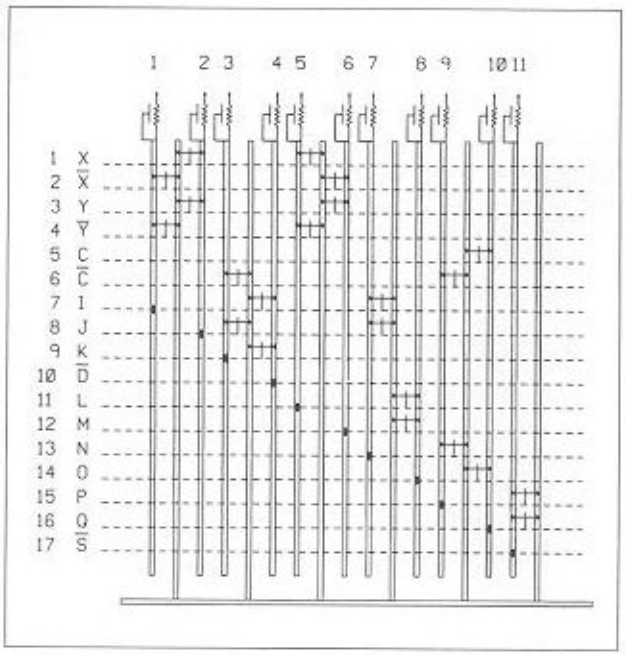

Figure 1. NOR n-MOS Weinberger array for expressions s and d.

```
01001000000
10000100000
01000100000
10001000000
00000000010
00100000100
20010010000
02100010000
00210000000
00020000000
00002001000
00000201000
00000020100
00000002010
00000000201
00000000021
```

Figure 2. Weinberger personality corresponding to Fig. 1.

To specify the exact function of the layout in a succinct way, we define a personality matrix for the Weinberger array. The array is represented as a matrix which contains 0s, 1s, and 2s. A '1' means that there is a transistor in the intersection of the column and the row, a '0' means no transistor in the intersection of a column and the row and a '2' means that there is a contact-cut between the column (vertical in metal) and the row (horizontal in polysilicon), and is also called the 'internal input'. The Weinberger personality (matrix$\{j, i\}$) corresponding to Fig. 1 is given in Fig. 2.

3. Column folding

The structure given in Fig. 1 corresponding to the personality in Fig. 2 can be modified to save area. This can be accomplished by *folding*. When we fold a WA, we allow the pull-ups to be placed both on the top and at the bottom of a column, and we hope that the transistors and output (contact-cut) corresponding to the first column will not overlap with those coming from the pull-up corresponding to the second column. Thus saving of area results when the circuits of two columns are put in the area that is required by one column in the unfolded array.

3.1. *WA column-folding algorithm*

A brief explanation is given below of the column-folding algorithm, which first orders the columns and then folds. More details of the algorithm and the data structure can be obtained from the detailed algorithms given in the last two figures.

The input to the algorithm is a WA personality, and the output is a folded WA with the maximum possible pairing of columns and a new ordering of rows.

The effect of the ordering algorithm in general is to reduce the usage of column area by putting the contact cuts as high as possible and to make the contact cuts run roughly diagonally downwards. Once this order is accomplished, folding is attempted. In order to fold any two columns 'X' and 'Y', for any column 'X', column 'Y' has to be on its right, since the contact cut for 'Y' must be lower than the contact cut for 'X'. Observe in Fig. 3 that it is possible to fold column 3 under column 2, which is on the right of column 2, but not vice versa. So also column 4 or column 2 can go under column 1 but the reverse is not possible. Thus, as the folding algorithm proceeds, the search space for columns is considerably reduced, resulting

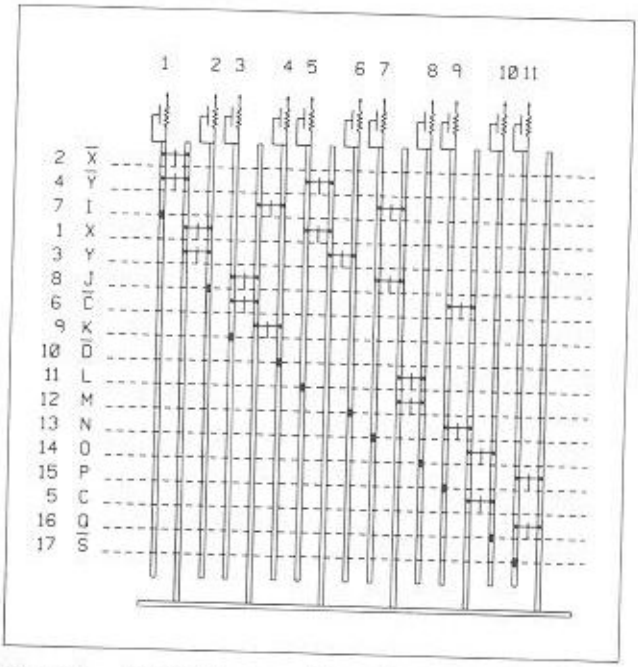

Figure 3. Weinberger array after column and row ordering.

in improved running times. A step by step procedure explaining the ordering of columns and rows is given below.

/* Order the columns and rows as follows */

Step 1. Repeat until all the rows are ordered.

Step 2. For each not ordered column *i* do.

Step 3. For each row *j* do.

Step 4. If (matrix[*j*, *i*] = '1').AND.(row *j* is internal input with '2' at column '*X*'). AND.(row *j* is not ordered yet) then skip the current column and go to the next column (column '*X*' has to be ordered *first*).

Step 5. Endfor (Step 3).

Step 6. If it is possible to order this column then order it next to the previous columns (or as the first column if there is no previous column).

/* Next order the rows as follows */

Step 7. For each row *j*, If (matrix[*j*, *i*] = '1').AND.(row *j* is not ordered), then put row *j* next to the previous ordered rows. (If there are no previous rows then row *j* will be the first row.)

Step 8. Endfor (Step 7).

Step 9. Now having processed all the rows in column '*i*' let the row with '2' be ordered next. (This step makes the '2' below all the '1s' in column '*i*' and makes the '2' in column '*i*' above the '2' of column '*j*' ordered after column '*i*').

Step 10. Endif (Step 6).

(Note: if all the rows have not been ordered and all the columns have passed without being considered for ordering, then there is a cycle in the input personality matrix, that is, some column, say '*X*', requires column '*Y*' to be ordered first and vice versa).

Step 11. Endfor (Step 2).

Step 12. End (repeat).

Steps 3–5 determine whether the column chosen in step 2 is a candidate to be in the ordered list of columns. Accordingly, rows in that column which have '1s' will be ordered in Steps 7 and 8. Steps 9 and 10 will put the row with a contact cut immediately below the lowest transistor in the selected column. This procedure gives a roughly diagonal ordering of contact cuts independent of the order of rows or columns in the initial array.

Having ordered the rows and columns, folding is attempted. A column '*Y*' (which is to the right of column '*X*') can be folded under column '*X*' if and only if the row of the topmost '1' in column '*Y*' is below or equal to the row of the '2' in column '*X*'. All the columns are processed in sequence from left to right to get the maximum possible folding using the following rules.

Rule 1

If a column has '1s' in rows corresponding to only external inputs then it is not to be folded.

Rule 2

If column '*Y*' has some '1s' in rows corresponding to internal inputs then try to fold it as follows:

(*a*) If the topmost '1' in column '*Y*' corresponds to internal input (row *j*) then we can fold column '*Y*' under column '*X*' which has a '2' in row *j* or any column to the left of '*X*'. Column '*X*' is the longest column under which column '*Y*' can be folded. So the following procedure is used.

(*a* 1) If column '*X*' is not used then fold '*Y*' under '*X*'.

(*a* 2) Otherwise, take the next column to the left of '*X*' (which is the next longest column under which '*Y*' can be folded) and try to fold '*Y*'.

(*a* 3) Repeat step (*a* 2) (that is, taking next column to left) until '*Y*' has been folded.

(*b*) If the topmost '1' in column '*Y*' corresponds to external input (row *i*), then check the column to the left of '*Y*'. If the '2' is in a row below row *i* then take the next column to the left and continue checking. Otherwise, try to fold column '*Y*' under that column or a column to its left if it is used (the same way as in part (*a*)).

It may be observed that if column '*Y*' can be folded under column '*X*', then we check that there is no column '*Z*' to the right of '*Y*' which can also be folded under '*X*'. If such a column '*Z*' exists, then it is folded under '*X*' because it is longer than '*Y*'.

4. Examples and concluding remarks

The WA obtained on application of the ORDER algorithm is given in Fig. 3, and the folded WA obtained on application of FOLD is given in Fig. 4.

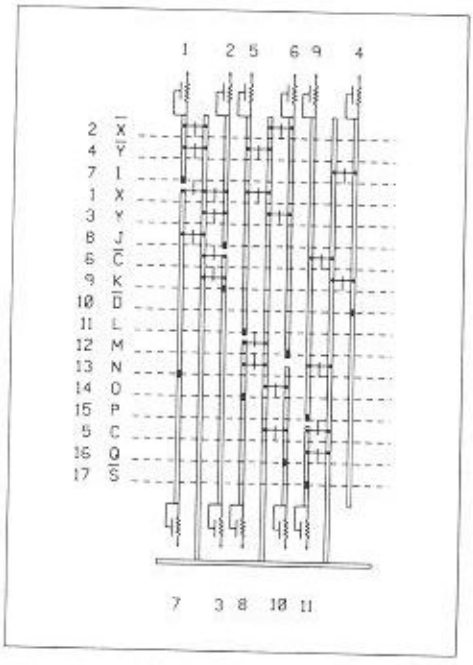

Figure 4. Folded Weinberger array of Fig. 3.

In Fig. 1, the starting array has a degree of order in it from the beginning, that is external inputs near top, and cuts running roughly diagonally downwards. This is not essential. The external inputs and other rows may be in any order and the algorithm still works adequately. This can also be observed in the second example in Fig. 5(*a*), where the '*' corresponds to a contact cut. It may be noted that each column has a '*' and they are not in a diagonal in the initial array.

From Figs 3 and 4, applying the rules of algorithms to the first four columns of the circuit in Fig. 3, we notice the following.

Column 2 uses two external inputs, but if we fold it under column 1 then columns 3 and 4 cannot be folded (violation of Rule 1).

Column 3 can be folded under column 1, but then column 4 cannot be folded. If we use rule 2(*a*) of §3 and fold column 3 under column 2, then column 4 can be folded under column 1. Since column 7 is longer than column 4 (the size of the column is the distance between the top transistor and the bottom-most transistor or contact cut in the ordered array) and the topmost 'I' is in the same row for both columns, then it is better to fold column 7 under column 1 because column 4 is shorter than column 7 and the chances of folding other columns under it are good.

Observe in Fig. 3 that column 2 has a contact cut in row 8 and column 3 has its topmost transistor in the same row. In order for the same 'grid point', that is, the intersection of folded columns 2 and 3 and row 8, to share both the contact cut and the transistor, the transistor has to be flipped. If this is not done then the topmost transistor in the second column (column 3 in this case) to be folded will have to be in a row below the contact cut of the first column (column 2). Thus, flipping of some transistors near the contact cuts, about their horizontal axis, serves as a means of expediting folding.

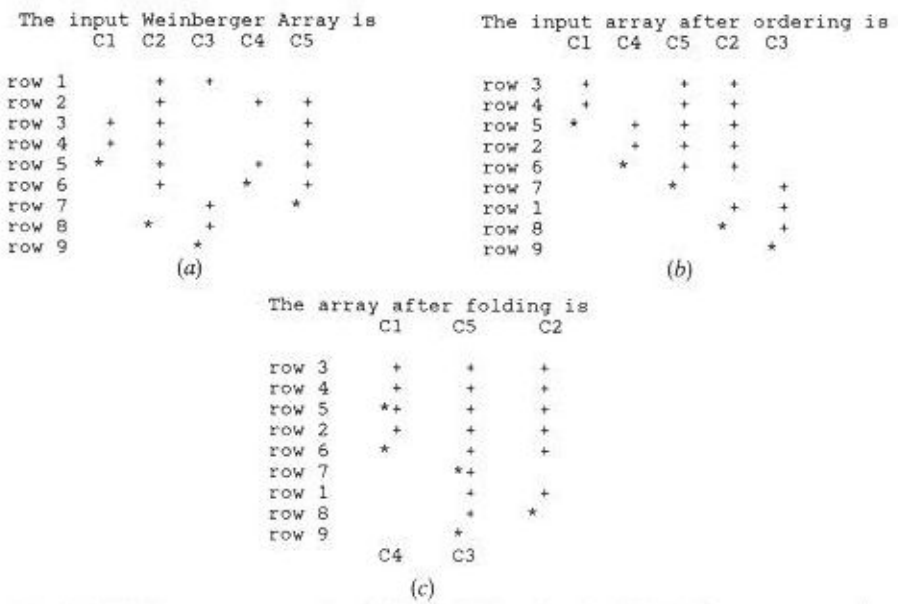

Figure 5. (*a*) Weinberger array to be folded with fan-in < 6. (*b*) Weinberger array after ordering. (*c*) Folded Weinberger array.

As another example to be folded, consider the array given in Fig. 5(*a*). The NOR gates in the array have a fan-in < 6 as can be seen by the number of 1s in any column. The column- and row-ordered array is given in Fig. 5(*b*) and the folded array is given in Fig. 5(*c*). In Figs 5(*a*)–(*c*), the '*' corresponds to the contact cut and '+' corresponds to a transistor. In Fig. 5(*b*) the rows and columns are ordered as shown. It is obvious from Fig. 5(*b*) that a single column can be used for the circuit in columns C1 and C4 and so also for the circuit in columns C5 and C3. The '*+' combination at the intersection of row 5 and C1 in Fig. 5(*c*) is the point of contact cut for C1 and a transistor for C4.

The algorithm was tested on several personalities. These included both practical circuits and randomly generated circuits. The constraints on the generation of random personalities (circuits) was on fan-in, number of inputs, number of outputs and the number of columns. The results of some randomly generated personalities are given in the Table. The maximum percentage saving of area by folding is 50%.

Fan-in	Ins	Outs	Cols B	Cols A	% Reduc
2	2	3	10	6	40·0
4	4	6	20	11	45·0
3	2	1	10	6	40·0
3	4	3	15	8	46·7
2	3	2	18	9	50·0
2	3	3	45	23	48·8
2	3	2	20	10	50·0
2	4	2	45	23	48·8
2	6	3	61	31	49·2

'Ins' is the number of inputs; 'Outs' is the number of outputs; 'Cols B' is the number of columns before folding; 'Cols A' is the number of columns after folding; '% Reduc' is the reduction in area due to folding.

It has been observed that the results are optimal in the sense that the pairing of columns is best and the algorithm folds two columns of the largest size, if such folding is possible.

The nearly pseudo-codes for the algorithms ORDER and FOLD have been explained in § 3. The actual algorithms given in Figs 6 and 7 give more implementation details and the data structure used and enable direct coding in any high-level language.

Column folding gives a maximum saving of 50%. Row compaction can then be performed to reduce the area further. For example the first row in Fig. 4 can be used for both X' (2) and d' (10) if the polysilicon running horizontally is broken between columns 9 and 4 and the output d' is taken from the right-hand side. With proper ordering of columns, excellent results can be obtained. An algorithm for row compaction has been given by Sait and Al-Khulaiwi (1990). One point to be noted is

```
Algorithm ORDER
Input : WA personality matrix
Output: An ordered matrix to be used by the FOLD algorithm
nr = # of rows
nc = # of columns
row[r] = rth row in the ordered WA
col[cl] = clth column in the ordered WA
matrix[j,i]= jth row, ith column in the input matrix
rowtocol[j]= the column at which row j has a 2 (if it has)
coltorow[i]= the row at which column i has a 2
BEGIN
   r=1, cl=1
   unmark all the rows and columns of the input matrix
   while (r<=nr) Do
      Begin
      For i = 1 to nc Do
         if column i in input  matrix is unmarked then
            Begin
               reject=false
               For j = 1 to nr Do
               Begin
                  If (matrix[j,i] = '1').and.
                    (j is internal input).and.
                    (row j in the input matrix is unmarked) THEN
                  reject=true'
               End
            If not reject THEN
               Begin
                  For j = 1 to nr Do
                  If (matrix [j,i] ='1').and.
                  (row j in input matrix is unmarked) THEN
                  Begin
                     mark row j in input matrix
                     row[r]=j
                     r = r+1
                  End
                  Else
                     If (matrix[j,i]=2) then
                        g=j
                     Endif
                  mark row g in input matrix
                  row[r]=g
                  r=r+1
                  col[cl]=i
                  cl=cl+1
                  rowtocol[g]=cl
                  coltorow[cl]=r
               End
            End
      End
ENDORDER
```

Figure 6. Pseudo-code for ORDER algorithm.

```
Algorithm FOLD
Input : An ordered WA
Output: A Folded WA
nr = # of rows
nc = # of columns
row[r] = rth row in the ordered WA
col[cl] = clth column in the ordered WA
rowtocol[j] = the column at which row j has a 2 (if it has)
coltorow[i] = the row at which column i has a 2
toptr[i] = top transistor (or top 1) in column i
tryfold[i] = Boolean function that is true only if column 'i'
             has a one at a row corresponding to internal input.
BEGIN
unmark all columns i
For i = 1 to nc Do
   If (i is unmarked) then
   Begin
      If (tryfold(i)) then
         Begin
            j = toptr[i]
            k = i
            For s = i+1 to nc do
               'If(toptr[s]=j).and.(tryfold(s))THEN
                  k=s
               If (row[j] is internal input) then
                  Begin
                     If (rowtocol[row[j]] is unmarked) then
                        Begin
                           Mark k and rowtocol[row[j]]
                           Fold col[k] under col[rowtocol[row[j]]]
                        End
                     Else
                        Begin
                           c=rowtocol[row[j]]-1
                           folded=false
                           while (c>=1).and.not(folded) do
                              if (c is marked) then
                                 c=c-1
                              Else
                                 folded=true
                           if folded then
                              Begin
                                 mark c and k
                                 fold col[k] under col[c]
                              End
                        End
                  End
               Else
                  Begin
                     c=k-1
                     folded=false
                     while (c>=1).and.not(folded) Do
                        If (j < coltorow[c]) then
                        c=c-1
                        else
                           if(c is marked) then
                              c=c-1
                           else
                              folded=true
                     if (folded) then
                        Begin
                           mark c and k
                           fold col[k] under col[c]
                        End
                  End
         End {tryfold}
End{for i}
ENDFOLD
```

Figure 7. Pseudo-code for FOLD algorithm.

that 50% reduction in the width of the array due to folding reduces more area on the wafer than 50% reduction due to compaction. This is because the minimum separation between two vertical metal lines (4λ) is greater than the minimum separation between two horizontal polysilicon lines (2λ) (Mead and Conway 1980).

The algorithms are coded in turbo Pascal and stick diagrams and layouts are produced on the Intergraph system. The Pascal code can be obtained by writing to the authors.

Acknowledgment

The authors acknowledge the support of the King Fahd University of Petroleum and Minerals. They also thank Mr Ahmed A. Al-Sheikh for his help in making the illustrations.

References

Ayres, R. F., 1983, *VLSI: Silicon Compiler and the Art of Automatic Chip Design* (New York: Prentice Hall).

Brayton, R. K., Hatchell, G. D., Hemachandra, L. A., Newton, A. R., and Sangiovanni-Vincentelli, A. L. M., 1982, A comparison of logic minimization strategies using EXPRESSO: an APL program package for partitioned logic minimization. *Proceedings of the I.E.E.E. International Conference on Circuits and Computers.*

Hemachandra, L. A., 1982, GRY: a PLA minimizer. Unpublished memorandum, Department of Computer Science, Stanford University, Stanford, California, U.S.A.

Mead, C., and Conway, L., 1980, *Introduction to VLSI Systems* (New York: Addison-Wesley).

Sait, S. M., and Al-Khulaiwi, F. A., 1990, Automatic Weinberger synthesis from a UAHPL description. *International Journal of Electronics*, **69**, 211.

Weinberger, A., 1967, Large scale integration of MOS complex logic: a layout method. *I.E.E.E. Journal of Solid State Circuits*, **2**, 182–190.